A Comparative Evaluation of Schema Subsetting for LLM-based NL-to-SQL over Large-Schema Databases

Kyle Luoma[*]
kyle.luoma@westpoint.edu
United States Military Academy - Army Cyber Institute
West Point, New York, USA

Arun Kumar
akk018@ucsd.edu
University of California, San Diego
La Jolla, California, USA

ABSTRACT

Large Language Models (LLMs) have become the standard for natural language interfaces to databases, but their effectiveness can be limited by context window constraints, especially for databases with large schemas. Schema subsetting or linking, which is the task of reducing the schema information provided to the LLM, has emerged as a strategy to address these limitations, yet its impact on NL-to-SQL performance remains unclear, particularly for very large schemas. In this paper, we systematically evaluate 7 real-world schema subsetting modules across 3 contemporary NL-to-SQL benchmarks, including Bird, Spider 2, and SNAILS, and we introduce BigBird–an expansion of the Bird benchmark datasets that provides additional data for evaluating subsetting of large schemas. We also introduce new subsetting-specific performance and efficiency metrics that enable in-depth evaluation of subsetting methods. Our analysis aligns with other recent work that suggests that most subsetting methods actually degrade NL-to-SQL execution accuracy from between 3% - 10% (model and method dependent) on smaller schemas, but also reveals that some subsetting methods can improve NL-to-SQL execution accuracy by up to 2% - 7% and others reduce token usage while generally maintaining the same execution accuracy performance as full-schema representations on large schemas. We also present SKALPEL, a prototype hybrid subsetting method that combines LLM-based question decomposition with semantic search, suggesting the potential for reduced token usage in NL-to-SQL workflows. These findings clarify the trade-offs of schema subsetting and motivate future research on scalable schema linking for large databases.

PVLDB Reference Format:
Kyle Luoma and Arun Kumar. A Comparative Evaluation of Schema Subsetting for LLM-based NL-to-SQL over Large-Schema Databases . PVLDB, 19(9): 2019 - 2031, 2026.
doi:10.14778/3819518.3819531

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/KyleLuoma/SKALPEL-subsetting-evaluation.

[*]This work was done in part while this author was a student at UC San Diego.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819531

1 INTRODUCTION

In the pursuit of better natural language interfaces with databases, Large Language Models (LLMs) have become the defacto standard for translating natural language questions into SQL queries. To avoid confabulating non-existent schema elements and ensure valid SQL identifiers, LLMs must include text-based schema knowledge in prompts. However, real-world databases, especially enterprise systems, often contain thousands of tables and tens of thousands of columns, making full schema representations exceed even some modern LLM context windows. In these cases, schema knowledge representations that include supplemental information such as column and table descriptions can exceed even SoTA LLM context window limitations, which can significantly degrade or eliminate the ability of an LLM to generate valid SQL queries.

In addition to the high token cost of including an entire schema reprsentation in prompts as well as the hard constraint of context window limitations, even the newest LLMs are still susceptible to degraded information recall due to large inputs. This tendency is commonly referred to as the *needle in a haystack* problem, and more recently as *context rot* [6] that describes a model's ability (or failure) to retrieve information from input prompts [7]. Although some recent large context models such as GPT4.1 claim to have successfully mitigated needle in a haystack-related errors, the ability of a model to perform more complex tasks more akin to determining the usefulness of a table or column to a natural language question, such as *multi-round co-reference resolution* (MRCR) [28], and semantic understanding of complicated tasks [6], are still negatively affected by high token counts [18].

The idea of reducing schema representation in LLM prompts to improve LLM-based SQL generation has taken form in several high-ranking NL-to-SQL methods on recent benchmark submissions. Some recent analysis of LLM-based schema linking methods provides a mixed view of the costs and benefits of such approaches. Our concurrent and complementary analysis of schema subsetting takes the body of schema research work even farther by asking two questions: 1) *what does the trade-off space of existing schema subsetting methods look like in terms of their impact on NL-to-SQL?* and 2) *how significant are its claimed benefits for large database schemas?*

Our Focus. In this paper, we extend the field of recent schema linking research by examining 7 real-world schema subsetting modules reproduced from NL-to-SQL project source code, and testing them over 3 benchmark datasets including Bird, the emergent Spider 2, and schema linking-focused SNAILS. We also address a shortage of NL-to-SQL examples over large schemas by creating an extension of the Bird benchmark dataset we call BigBird that increases the size of the database schemas in the Bird dev dataset. This is the first work to evaluate real-world schema linking modules, using schema linking-focused performance and efficiency metrics, over very large database schemas. Using the lessons learned from our benchmarking and analyses, we present a prototype hybrid schema linking method we call SKALPEL to motivate additional research toward more effective and efficient schema linking.

Schema Subsetting (AKA Schema Linking). Schema subsetting (also known as schema filtering, schema linking, structured grounding, or entity retrieval) is an approach used by many database NLIs to reduce schema knowledge size as a method for avoiding context window limitations and mitigating the needle in the haystack problem that can occur with very large prompts. It also serves to reduce NL-to-SQL inference cost by reducing token counts in LLM prompts. With LLM-based NL-to-SQL launching the usefulness of natural language interfaces (NLI) to databases into the realm of the practical, attempts to further improve the quality of SQL inference have begun to focus on sub-problems including schema linking. Recent analysis yields mixed results. On one hand, oracle-based evaluation suggests that reducing false positives improves NL-to-SQL generation for smaller LLMs [8]. On the other hand, real-world schema linking—without oracle knowledge—often proves detrimental to generation quality. This is primarily the case for the highest performing “flagship” LLMs [16]. Current schema linking methods and analysis has relied on the Bird [12] and Spider [31] benchmarks which have schema sizes smaller than many real-world systems.

Contributions. This paper makes the following contributions:

- A comprehensive benchmarking framework specifically aimed at the schema subsetting component of end-to-end NL-to-SQL workflows.
- A reproduction and empirical analysis of 7 real-world schema subsetting modules from published code repositories on 3 benchmark datasets: Bird, SNAILS, and Spider 2, and an extension of recent schema subsetting research to very large schemas.
- BigBird: an expansion of the schemas in the Bird benchmark dev dataset to enable more indepth research into NL-to-SQL over very large schemas.
- We present SKALPEL: a new subsetting scheme for improving NL-to-SQL efficiency over very large schemas.

2 BACKGROUND AND PRELIMINARIES

Schema knowledge subsetting (also known as schema linking) is the process of extracting a subset of relations and attributes from a database schema with the objective of eliminating as many unneeded identifiers while retaining all identifiers required for a correct query generation.

In this section, we review recent subsetting modules in NL-to-SQL systems ranked highly on the Bird benchmark. We measure subsetting effectiveness by ablating schema knowledge representations from a full schema to only identifiers present in a correct query, and we establish symbolic definitions of the subsetting problem.

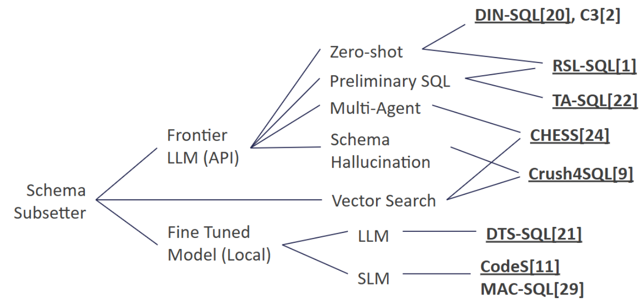

Figure 1: Schema subsetting methods use a variety of techniques and resources including frontier (API-based) LLMs, smaller locally-hosted and finetuned models (both LLM and SLM), and vector search. Some methods use more than one method such as CHESS which uses multiple LLM agents and vector search. Underlined methods indicate that a method is included in our experiments.

2.1 Survey of Subsetting Approaches

Many leading submissions to the Bird NL-to-SQL benchmark [12], including the top performer, use schema subsetting. As of June 2025, 5 out of the top 15 entries [3, 4, 13, 23, 24, 30] on Bird’s execution accuracy leaderboard use some form of schema subsetting in their SQL generation pipelines. While some analysis suggests that the risk of omitting required identifiers during the subsetting step negates any potential benefit [16], the overall performance of the top submissions that use schema subsetting suggest that, when done correctly, subsetting may be improving NL-to-SQL performance in some cases. These mixed results motivate additional scrutiny of existing schema subsetting methods to 1) determine their overall usefulness for improving NL-to-SQL, and 2) discover specific aspects of subsetting methods that either improve or degrade NL-to-SQL generation.

Selection Criteria for Evalution. Our research objective is to evaluate the best representations of a variety of subsetting methods, most of whose relatively high positions on the Bird execution accuracy leaderboard indicates good performance. Our ability to reproduce a given subsetting method is limited by code and model availability, and this constraint eliminates a large proportion of entries, as only 34 of the 74 entries provide links to their source code. Of the remaining entries, we select entries that indicate the use of a schema subsetting step and provide a fully reproducible codebase and, where applicable, model weights. Figure 1 shows the 7 subsetting methods we evaluate in our research and their classification in terms of subsetting model type and technique.

Subsetting with LLMs. Several approaches make use of the same LLM used for NL-to-SQL generation in a prior step that includes submitting a full schema, a natural language question, and instructions to identify the most-relevant schema entities. The output of this reduction step is typically used as input to an SQL generation prompt. LLM-based subsetters are subject to the same context limit constraints as an SQL generation prompt, and so a database schema representation that is too large for naive NL-to-SQL inference with full schema knowledge will also be too large for a schema subsetting prompt.

DIN-SQL [20], and C3 [2] use LLM-based schema subsetters as preliminary actions in a multi-step zero-shot NL-to-SQL generation pipelines. RSL-SQL [1] uses an LLM-based bi-directional subsetting method that combines the results of two actions: 1) a few shot prompt instructing the LLM to select relevant schema elements based on a natural language question, and 2) and extraction of table and column names from a *preliminary SQL* query generated via zero-shot prompting and the full schema representation. TA-SQL [22] pre-processes a database with LLM-generated column descriptions which are used to supplement schema knowledge representations in a zero-shot SQL generation prompt. TA-SQL generates subsets through identifier extraction from the generated preliminary SQL queries.

Hybrid Subsetting Methods. Some methods employ multiple techniques including the use of both LLMs and semantic search. CHESS [24] is an agentic system that contains information retriever and schema selector agents that execute their tasks sequentially to generate schema subsets. The information retriever agent has access to information retrieval tools that use vector searches over identifier embeddings to retrieve the most relevant identifiers. The schema selector agent refines the information retriever's selections using LLM-based few shot prompting. Crush [9] is the only method that does not correspond to a Bird benchmark entry. It is an LLM-based subsetter that introduces the novel idea of schema hallucination as a pre-step to semantic search of vector representations of the target database's identifiers.

Subsetting with Finetuned Models. Another approach is to finetune a smaller language model (SLM) specifically to the task of schema subsetting. DTS-SQL [21] uses a finetuned 7B parameter Deepseek Coder [5] to subset schemas by selecting relevant tables. Code-S [11] employs a pre-trained T5-based model and adds a value lookup feature that seeks alignment between natural language symbols and text values in a target database. Code-S employs subsetting-specific analysis via area under curve (AUC) metrics for table and column recall. The authors of DTS-SQL introduce the use of precision and recall to evaluate schema linking performance separately from overall NL-to-SQL performance. MAC-SQL [29] uses a selector agent composed of an LLM prompt sequence run against a finetuned 7B parameter LLM to reduce a database to a subset database aligned with an input natural language question.

2.2 Subsetting Effectiveness

Schema Subset Proportion Effects on NL-to-SQL. To test the assumption that schema subsetting improves NL-to-SQL performance, we perform subset proportion ablation where we modify schema representations iteratively to randomly reduce the proportion of unneeded schema identifiers in a SQL creation prompt, where a proportion of 1.0 contains all schema identifiers, and a proportion of 0.0 contains only the identifiers required for a correct result. Figure 2 shows the result of a zero-shot prompt with varying proportions of schema knowledge executed over the SNAILS benchmark dataset at the Native naturalness level. There is an evident

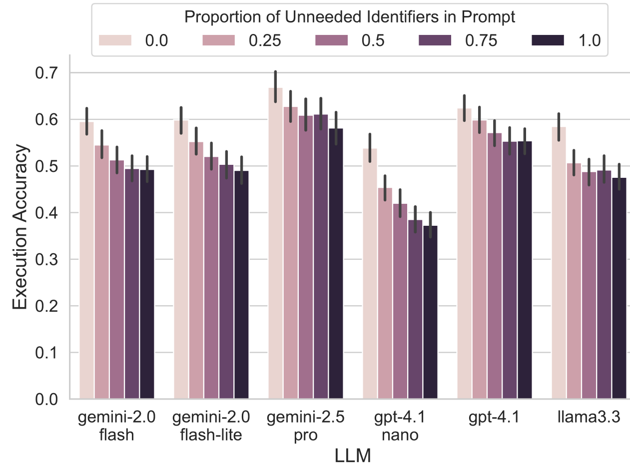

Figure 2: Execution accuracy (y-axis) over the medium- and large-sized schemas in the SNAILS benchmark generally exhibits a downward trend as the proportion of unneeded identifiers increases.

trend of performance improvement as the proportion of unneeded identifiers decreases for all models at most proportion levels. When the schema representation sheds all unneeded identifiers (the case of a perfect subset), execution accuracy improves dramatically for all models.

The takeaway from this evaluation is two-fold: 1) the usefulness of schema subsetting at different proportions is model-dependent, and 2) the potential for performance improvement in the case of a subset with perfect recall motivates further evaluation and improvement of real-world subsetting methods.

2.3 Definitions and Notation

Schema and Database. We evaluate subsetting processes over a relational database schema $\mathbb{S}$ that contains a set of relations $\mathcal{R}$ and their attributes $\mathcal{A} \in \mathcal{R}$. Each relation $\mathcal{R}$ is a bag of tuples $t_{1\ldots n}$ where $t[\mathcal{A}]$ references an atomic value v of the type of attribute $\mathcal{A}$. The database instance $\mathbb{D}$ is the realization of $\mathbb{S}$ such that $t[\mathcal{A}_{1\ldots n}] \in \mathcal{R}_{1\ldots n} \in \mathbb{S} \Rightarrow \mathbb{D}$, and an SQL query Q_{sql} over $\mathbb{D}$ returns a result r which is also a bag of tuples. We denote the set of all values v in $\mathbb{D}$ as $\mathbf{V}$, and specify the set of all values in a given relation or a relation and attribute as $\mathbf{V}_{\mathcal{R}}$ and $\mathbf{V}_{\mathcal{R}.\mathcal{A}}$ respectively.

SQL Queries and Identifiers. An SQL query Q_{sql} references relations $\mathcal{R}$ and attributes $\mathcal{A}$ in its projection and selection clauses. We differentiate between SQL queries that are members of benchmark question and query pairs as "gold" queries, with the notation $Q_{sqlGold}$ and queries that are predicted by an NL-to-SQL function with the notation $Q_{sqlPred}$. For the purposes of subsetting analysis, we consider Q_{sql} in a syntax-agnostic manner, simply as a set of its identifiers organized as a schema subset $\mathbb{S}'$ containing a subset of schema relations $\mathcal{R}'$ and attributes $\mathcal{A}'$.

We also define selection clause predicates $\mathcal{P}$ in simple terms of an attribute $\mathcal{A}$ and value v, and denote them in the form $\mathcal{P} :=< \mathcal{A}, v >$ where for a given predicate $\mathcal{P}$, $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_v$ are the attribute and

Table 1: Schema size distribution for each benchmark dataset.

Size	Columns	BigBird	Bird	Snails	Spider2	Total
S	<100	0	9	1	34	44
M	<1,000	0	2	6	43	51
L	<2,500	0	0	1	10	11
XL	<50,000	11	0	0	14	25
XXL	<100,000	0	0	1	4	5

value referenced in the clause. For example,
$\mathcal{P} :=< customer.name, \text{"}jones\text{"} >$ represents the selection expression $customer.name = \text{"}jones\text{"}$.

Benchmarks and Natural Language Question Symbols. A natural language question Q_{nl} is comprised of an ordered sequence of symbols $w \in Q_{nl}$ where symbols may be words, numbers, or special characters.

NL-to-SQL benchmarks $\mathbb{B}$ are collections of databases $\mathbb{D}$ and associated natural language question Q_{nl} and gold SQL query $Q_{sqlGold}$ pairs. We denote each pair of $Q_{nl}, Q_{sqlGold} \in \mathbb{D} \in \mathbb{B}$ as $\mathbf{Q}$ so that $\mathbf{Q} :=< Q_{nl}, Q_{sqlGold} >$.

Subsetting Function. We define an abstract schema subsetting function Ψ which generates a schema subset $\mathbb{S}'$ given the inputs Q_{nl} and $\mathbb{D}$. This function uses an abstract comparison operation denoted as $\sim$ (and its negation $\nsim$) which is satisfied if there is sufficient semantic similarity or other relevance-defining relationship between one or more symbols in Q_{nl} and one or more entities in a database $\mathbb{D}$.

$$\Psi(Q_{nl}, \mathbb{D}) \rightarrow \{\mathbb{S}' | \mathbb{S}' \subseteq \mathbb{S}\}$$

where relations and attributes are included in the subset by the criteria:

$$\Psi(Q_{nl}, \mathbb{D}) := \forall w \in Q_{nl} \exists \mathcal{R} | \mathcal{R} \in \mathbb{D} \wedge (\mathcal{R} \sim w \vee (\exists \mathcal{A} \in \mathcal{R} | \mathcal{A} \sim w)) \quad (1)$$

That is, relations and attributes are members of the schema subset $\mathbb{S}'$ if they satisfy the semantic similarity comparison operation $\sim$ for at least one symbol w in Q_{nl}. Note that there is no restriction on the $\sim Q_{nl}$ operation so that we can overload Ψ and its $\sim$ operator in specific implementations in subsequent sections.

3 METHODOLOGY

To thoroughly evaluate multiple subsetting methods across multiple benchmarks, we create a modular object-oriented architecture to standardize interfaces between benchmarks, subsetting methods, evaluation processes, and NL-to-SQL generation. The core of this approach is the schema data object, represented in our project as a Python object that can be transformed to DDL and JSON representations and is exclusively used as the representation of both full and subset schemas throughout the project codebase.

3.1 Benchmarks

NL-to-SQL Benchmarks. The availability and difficulty of NL-to-SQL benchmarks have evolved at a pace relatively aligned with the advancement of NL-to-SQL systems. BIRD [31] is currently the most active benchmark, and offers a level of complexity that has posed a challenge for the current generation of LLMs and systems to achieve human-level performance. The BIRD benchmark consists of 95 databases, over 12,000 question-SQL pairs, and 37 domains. These data are split across a training, dev, and test set with the training and dev set available to the public.

Spider [31] was the most active NL-to-SQL benchmark at the beginning of the LLM-based NL-to-SQL era, and provides an interesting view of the associated advancement in NL-to-SQL capability over time. Spider has since reached benchmark saturation, and Spider 2.0 [10] supersedes it as the newest NL-based database interaction benchmark, focusing on multi-step data retrieval over large and complex schemas. Unlike its predecessors, Spider 2.0 does not provide dev or test set question-SQL pair data for model training. Instead, the authors make a subset of the test data question-SQL pairs available (256 total) to assist developers with prompt design. Spider 2.0 extends the set of target databases beyond Sqlite and also includes both Snowflake and BigQuery databases.

The SNAILS [14] project contains a benchmark dataset of 9 databases and 503 question-SQL pairs designed to represent real-world relational database schema design patterns including schema naming practices (defined as naturalness), schema size, and complex dependencies.

BigBird: An Extension of the Bird Dataset. An obstacle to achieving the goal of evaluating subsetting over large schemas is the limited number of $\mathbf{Q}$ aligned to large schemas in existing NL-to-SQL benchmark datasets. Although the Spider2 datasets contain multiple examples of large schemas, the availability of $\mathbf{Q}$ associated with these schemas is insufficient for a meaningful evaluation. To remedy this shortage, we take advantage of the Bird [12] dev set consisting of over 1,500 $\mathbf{Q}$ by supplementing the 11 $\mathbb{D}$ in the $\mathbb{B}$ with additional $\mathcal{R}$ and $\mathcal{A}$ elevating them to the XL category (see Table 1).

We create the additional schema objects by prompting the GPT-OSS-120b LLM with a schema represenation containing the original tables as well as additional tables created in prior iterations. We provide the instruction to create new tables that extend the semantics of the schema within its domain without duplicating existing information. This avoids semantic collisions with the base tables targeted by $\mathbf{Q}$ so that the $Q_{sqlGold}$ does not change given the larger schemas. Details of this implementation are available in the technical report [15].

Benchmark Data. To evalute subsetting methods against a diverse set of schemas, we retrofit the Spider2 [10], SNAILS [14], Bird [12], and BigBird NL-to-SQL benchmarks. All benchmarks conform to the $\mathbf{Q} :=< Q_{nl}, Q_{sqlGold} >$ format, where NL questions are paired with a ground truth $Q_{sqlGold}$ used for evaluating correctness of subsetting method-generated $Q_{sqlPred}$ queries.

Table 1 summarizes the benchmark schemas in terms of 5 size categories based on the total number of columns in a schema. The Bird dev set, the benchmark that appears most often in similar research, contains only small- and medium-sized schemas whereas the SNAILS and Spider 2 collections contain much larger schemas. The BigBird benchmark extension expands the availability of extra large-sized schemas. Schema size categorization enables a cross-benchmark evaluation of subsetting that prioritizes overall schema size over the source benchmark, which is important in subsetting evaluation because some methods are constrained by model input context limits that may render them useless in a particular size category.

Benchmark Integration. The four benchmarks differ from each other in terms of how they perform schema representation, gold query and natural language question pairs formatting, and database storage and access. We define an abstract NL to SQL object with standard function calls and iterators that serves as the super class to the individual benchmark sub-classes that wrap the underlying benchmark behaviors in a standardized interface. This approach allows us to perform online benchmark instantiation using a factory class and greatly increases the interchangeability of benchmarks in the subsetting evaluation and NL-to-SQL experiments.

3.2 Schema Subsetting

We modularize and abstract the schema subsetters in a similar fashion as the benchmark class. Extracting the subsetting behavior from real-world NL-to-SQL systems varies in complexity by each system with some methods requiring more complex interfaces than others. Regardless of underlying method complexity, we are able to apply a singular schema subsetter interface to each one, thus also making the subsetter modular and relatively easy to modify existing methods and experiment with new methods. Schema subsetters accept 3 inputs: a benchmark question object, an associated schema object, and a natural language question. They return a result that contains a schema object (subset) and token usage data.

3.3 NL-to-SQL Generation

There are a variety of approaches to LLM-based NL-to-SQL generation, and the NL-to-SQL systems from our evaluated subsetting modules employ various post-subsetting techniques including zero-shot NL-to-SQL generation, error correction, few-shot example-based NL-to-SQL generation, etc. To maintain consistency in evaluation between subsetting methods, we forego more complex NL-to-SQL techniques and instead use a zero-shot prompting approach that includes SQL generation instructions, the schema subset (represented serially as table: column 1, column 2, ..., column n), and additional evidence or hints relating to the schema if provided in the benchmark dataset.

We evaluate 7 LLMs from the OpenAI GPT [17, 19], Meta Llama [26], and Google Gemini [25] model families. We categorize the models into 3 classes: *Flagship models* (GPT 4.1, Gemini 2.5 Pro), *economy models* (GPT 4.1-Nano, Gemini 2.5 Flash, Gemini 2.5 Flash-Lite), and *open source models* (Llama 3.3 70B Q8, GPT OSS 120B).

We use the OpenAI and Google API libraries for flagship and economy model integration. We host Llama 3.3 using Ollama on a dedicated server equipped with 2 AMD 9254 24-core processors, 541GB of system memory, and 2 NVIDIA A100 GPUs, with the model occupying 1 GPU. We host GPT OSS 120B using vLLM on a dedicated server equipped with 2 AMD 9654 96-core processors, 1,623 GB of system memory, and 8 NVIDIA H100 GPUs, with the model occupying 4 GPUs.

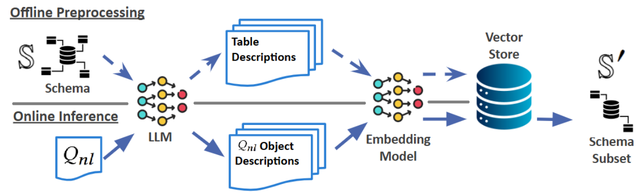

Figure 3: The prototype SKALPEL subsetter combines LLM-based question decomposition and embedding vector generation to perform table retrieval from a vector store. SKALPEL measures cosine distance between the embeddings of 2-3 sentence table descriptions and a set of object descriptions decomposed from a Q_{nl} and returns $\mathbb{S}'_{Pred}$ tables with all of their columns whose distance are below a specified threshold.

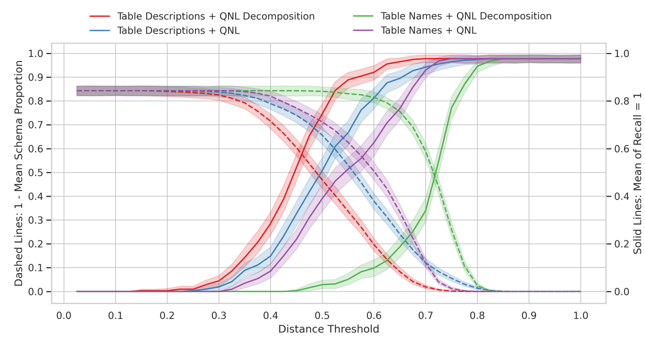

Figure 4: We compare perfect recall and schema proportion (inverted on Y-axis) by ablating the SKALPEL question decomposition and table description methods. We find that question decomposition and table description distances at a distance threshold of 0.6 yields a target combination of perfect recall (0.92) and schema proportion (0.80) which favors recall without sacrificing some potential benefit from schema size reduction.

3.4 The SKALPEL Subsetter

In addition to a thorough evaluation of existing subsetting modules, we evalute the potential of a hybrid semantic search and LLM-based subsetting method with the goals of reducing token usage and improving NL-to-SQL for very large database schemas (Figure 3). The primary design goal of SKALPEL is to maximize table and column recall while minimizing subset size, minimizing token usage, and reducing latency.

To accomplish this goal, we adopt a hybrid approach similar to Crush4SQL [9] except instead of schema hallucination, we task the LLM to decompose the question (similarly to the decomposition method described in [8]) with the instruction "describe the various objects, concepts, etc. in a natural language query." The output of the object description task is embedded using the NovaSearch *stella_en_1.5B_v5* embedding generation model [32].

During an offline preprocessing stage, SKALPEL uses an LLM to generate natural language descriptions of schema tables using the

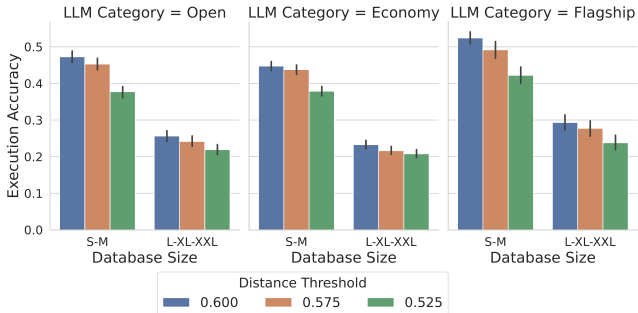

Figure 5: We evaluate NL-to-SQL performance over three cosine distance thresholds, using the question decomposition and natural language table description comparison approach. The y-axis (execution accuracy) suggests that the higher 0.600 distance threshold yields the best NL-to-SQL performance.

table name, column names, and example values. These descriptions are also embedded using NovaSearch and stored in a PGVector [27] database.

During subset generation, SKALPEL retrieves relevant tables and all their columns via cosine distance search, where the distance between question decomposition embeddings and table description embeddings are less than or equal to 0.600.

We select the threshold 0.600 from the results of an experiment which exposes the effects of threshold selection and feature ablation on recall and schema proportion (or subset size) (see Figure 4) and NL-to-SQL execution accuracy (see Figure 5).

During distance sensitivity analysis, we evaluate changes in perfect recall, which is the mean of an identify function $pr(r) = 1\ if\ r = 1,\ and\ pr(r) = 0\ otherwise$ where r is the recall score calculated from the evaluation of $\Psi(Q_{nl}, \mathbb{D}) \rightarrow \mathbb{S}'_{Pred}$ for all $\mathbf{Q} \in \mathbb{B}$.

In addition to distance sensitivity, we ablate SKALPEL by removing both question decomposition and NL table descriptions, and evaluating combinations of searches over raw table names and unmodified Q_{nl}. After the initial similarity distance sensitivity and feature ablation, we select 3 distances thresholds (0.525, 0.575, and 0.600) and run NL-to-SQL over all $\mathbf{Q} \in \mathbb{B}$ for all benchmarks. Figure 5 shows that the highest threshold evaluated (0.600) achieves the highest execution accuracy for all LLM categories, and so we select this threshold for all further evaluations of SKALPEL. Additional ablation study details are available in the technical report [15].

4 EXPERIMENTS: SUBSETTING PERFORMANCE

4.1 Discovering Schema Size Limits

In order to discover the limits of subsetting methods executed over large schemas, we assess subsetting method capabilities in the context of schema size categories (described in Table 1). We do this by determining the percent of subsetting attempts that yield a non-empty set of schema identifiers, which indicates that the subsetting method did not encounter a condition that prevented it from generating a subset.

Table 2: Percent of each schema size category that each subsetting method is capable of processing. Only 3 of the 7 evaluated methods are capable of processing all schemas.

Ψ	S	M	L	XL	XXL
CodeS	100%	100%	100%	100%	100%
DINSQL	100%	100%	100%	100%	100%
CHESS	100%	100%	100%	-	-
Crush	100%	100%	100%	100%	100%
DTSSQL	66%	12%	-	-	-
RSLSQL	100%	100%	82%	88%	-
SKALPEL	100%	100%	100%	100%	100%
TASQL	100%	100%	100%	92%	-

Size Constraints. Exposing subsetting methods built to solve smaller schemas in the Bird benchmark to larger schemas in the SNAILS and Spider 2 benchmarks reveals some limitations in most of the subsetting methods. Methods that load the entire schema representation into a single LLM prompt tend to exceed LLM context window limitations on very large schemas. Only 3 of the 7 methods that we evaluate successfully completing inference over all benchmark schemas. Table 2 reveals the inference completion percentages for each model and schema size. CodeS, a machine-learning based classifier approach, handles its small 512 token context via partitioning. Crush4SQL is not subject to context window constraints because it performs table and column retrieval using semantic search. DINSQL generates a minimal schema representation without additional value examples or identifier descriptions and thus does not exceed the GPT4.1 context window. Chess did not exceed any context limitations, however it relies on a column-by-column assessment to determine column relevance–a method that proves to be prohibitively expensive and time consuming for very large schemas. In the following experiments, we only compare results for subsetting attempts that do not exceed a subsetting method's capabilities.

4.2 Setup and Metrics

We evaluate 7 subsetting methods used by high-performing NL-to-SQL submissions on the Bird [12] benchmark leaderboard. These methods represent a variety of approaches including frontier LLM prompting, SLM finetuning, and semantic search. We go beyond the standard measures of precision, recall, f1, and also include token usage (as applicable), inference time, preprocessing time (as applicable), and subset size as a proportion of the full schema. This allows us to assess the time and resource requirements for each subsetting method, and motivates the development of more economical methods.

Reproducing Subsetting Methods. We reproduce each of the 7 subsetting methods by integrating their publicly-available code into our experiment's modular evaluation architecture. Each method requires the development of adapters to ensure a common and consistent representation of $\mathbb{S}$ and $\mathbf{Q}$ in each benchmark $\mathbb{B}$. Each LLM-based and hybrid method requires API key configuration through global variables, configuration files, or embedded code. Some older methods (e.g., DINSQL) use deprecated LLMs. For consistency and fair comparison between approaches, we configure the LLM-based methods (DINSQL, TASQL, and RSLSQL) to use GPT 4.1.

Some subsetting methods, including hybrid methods that use semantic search, use benchmark schema data pre-processing, which we execute and evaluate in terms of time required. For small language model-based methods including CodeS and DTSSQL, we self-deploy the provided models, selecting the weights that indicated the best performance in their respective publications. Due to the complexity of such integrations, we leave the remainder of the technical details including hardware resource requirements, codebase modifications for experiment integration, and LLM service configurations to the technical report [15].

Research Question. How do subsetting methods compare in the trade-off space betwen recall, precision, f1, time-based performance, and resource usage?

Evaluation Method. We evaluate subsetting performance over the NL-SQL question-query pairs $\mathbf{Q} \in \mathbb{B}$ in the Bird, SNAILS, and Spider 2 benchmarks. The correct subsets used to derive recall, precision, and f1 scores include all identifiers (relations and attributes) present in each pair's gold SQL representation $Q_{sqlGold}$.

In order to derive the correct subset for each benchmark question, we define an SQL parsing function $P(Q_{sql}) \rightarrow \mathbb{S}'$ that arranges relations and attributes into a schema subset $\mathbb{S}'$ comprised of identifiers referenced in the input query Q_{sql}. For each $\mathbf{Q}$ in a benchmark $\mathbb{B}$, we derive a gold schema subset $\mathbb{S}'_{Gold}$ using $P(Q_{sqlGold}) \rightarrow \mathbb{S}'_{Gold}$. Because $Q_{sqlGold}$ is a syntactically correct query over its target $\mathbb{D} \in \mathbb{B}$, we assert that all $\mathbb{S}'_{Gold} \subseteq \mathbb{S}$ and that $\mathbb{S}'_{Gold}$ represents the execution of a perfect (or oracle) subsetter $\Psi(Q_{nl}, \mathbb{D}) \rightarrow \mathbb{S}'$. With this assertion, we evaluate the output $\mathbb{S}'_{Pred}$ of each subsetting method's implementation of Ψ against $\mathbb{S}'_{Gold}$ for each $\mathbf{Q}$.

$$\forall Q_{nl}, Q_{sqlGold} \in \mathbf{Q} \in \mathbf{B} : \quad \Psi(Q_{nl}, \mathbb{D}) \rightarrow \mathbb{S}'_{pred}; P(Q_{sqlGold}) \rightarrow \mathbb{S}'_{Gold} \qquad (2)$$

The end state of the process is a collection of predicted and gold subsets for all question-query pairs used as the input to the analysis described in the following sections.

Evaluation Metrics. Schema linking-specific metrics have begun to appear in recent NL-to-SQL papers including precision and recall [8, 9, 14, 16, 21], and AUC [11]. To the best of our knowledge, our work is the first application of schema linking metrics to a comparison of multiple reproductions of schema linking modules used in real-world NL-to-SQL systems. Additionally, we believe this to be the first work to evaluate schema subsetting in terms of efficiency by measuring token usage, offline pre-processing time, and inference time.

We evaluate performance of Ψ using the following metrics.

- SchRecall: $|\mathbb{S}'_{Gold} \cap \mathbb{S}'_{Pred}| / |\mathbb{S}'_{Gold}|$
- PerfRecall: 1 if *ScheRecall* = 1.0, 0 otherwise
- SchPrecision: $|\mathbb{S}'_{Gold} \cap \mathbb{S}'_{Pred}| / |\mathbb{S}'_{Pred}|$
- SchF1: $2 * (SchRecall * SchPrecision) / (SchRecall + SchPrecision)$
- Subset Proportion: $|\mathbb{S}'_{Pred}| / |\mathbb{S}|$; (lower is better)
- Inference Time: Mean time (in seconds) to execute Ψ over $\mathbf{Q} \in \mathbb{B}$.
- Prompt Token Count (LLM-based): Mean count of LLM tokens used by Ψ to generate $\mathbb{S}'_{Pred}$ for each $\mathbf{Q} \in \mathbb{B}$
- Preprocessing Rate: $SchemaProcessingTime / |\mathcal{A} \in \mathbb{S}|$ where *SchemaProcessingTime* is the time in seconds required to perform pre-processing tasks such as generating vector embeddings or data descriptions.

We expand the evaluation framework further by generating each of the listed metrics for each identifier type (relations and attributes).

- SchRelRecall: $|\mathcal{R}'_{Gold} \cap \mathcal{R}'_{Pred}| / |\mathcal{R}'_{Gold}|$
- SchRelPrecision: $|\mathcal{R}'_{Gold} \cap \mathcal{R}'_{Pred}| / |\mathcal{R}'_{Pred}|$
- SchRelF1: $2 * (SchRelRecall * SchRelPrecision) / (SchRelRecall + SchRelPrecision)$
- Relation Proportion: $|\mathcal{R}'_{Pred}| / |\mathcal{R} \in \mathbb{S}|$; (lower is better)
- SchAtrRecall: $|\mathcal{A}'_{Gold} \cap \mathcal{A}'_{Pred}| / |\mathcal{A}'_{Gold}|$
- SchAtrPrecision: $|\mathcal{A}'_{Gold} \cap \mathcal{A}'_{Pred}| / |\mathcal{A}'_{Pred}|$
- SchAtrF1: $2 * (SchAtrRecall * SchAtrPrecision) / (SchAtrRecall + SchAtrPrecision)$
- Attribute Proportion: $|\mathcal{A}'_{Pred}| / |\mathcal{A} \in \mathbb{S}|$; (lower is better)

Performance Constraints. While variance in precision indicate method quality, other metrics indicate method feasibility where failure to meet specific constraints will result in NL-to-SQL translation failure. That is not to say that a method should be discounted outright in the event of a failure to meet the constraint in some cases; rather the frequency of failures should be compared between methods with the best methods being those with the fewest failure occurrences.

Perfect recall: Failure of the subsetting function Ψ to recall all $\mathcal{A}'_{Gold}$ and $\mathcal{R}'_{Gold}$ in $Q_{sqlGold}$ will almost always ensure failure in the downstream NL-to-SQL translation task. Thus, it is imperative that we evaluate recall in the strictest terms possible with the goal of identifying methods that yield recall values as close to 1 as possible. The proportion of subsetting attempts that achieve perfect recall over all attempts represents the likely upper bound of execution accuracy in a subsequent NL-to-SQL step. Subsetting methods that achieve perfect recall scores below downstream NL-to-SQL (sans subsetting) execution accuracy proportion scores are likely to reduce overall performance.

Subset size: schema subsets must be sufficiently small that generated prompts fit within the target LLM's context window token limitations. Subsetting attempts that fail due to context window constraints are omitted from analysis. Subsetting method schema coverage (the percentage of schemas each method is capable of handling) is measured in Table 2.

4.3 Subsetting Evaluation Results

Table 3 displays all subsetting performance metrics for each method at each schema size where only successful subsetting attempts for each method are included in the evaluation calculations. Figure 6 provides a visual comparison of subsetting methods over key accuracy and performance metrics.

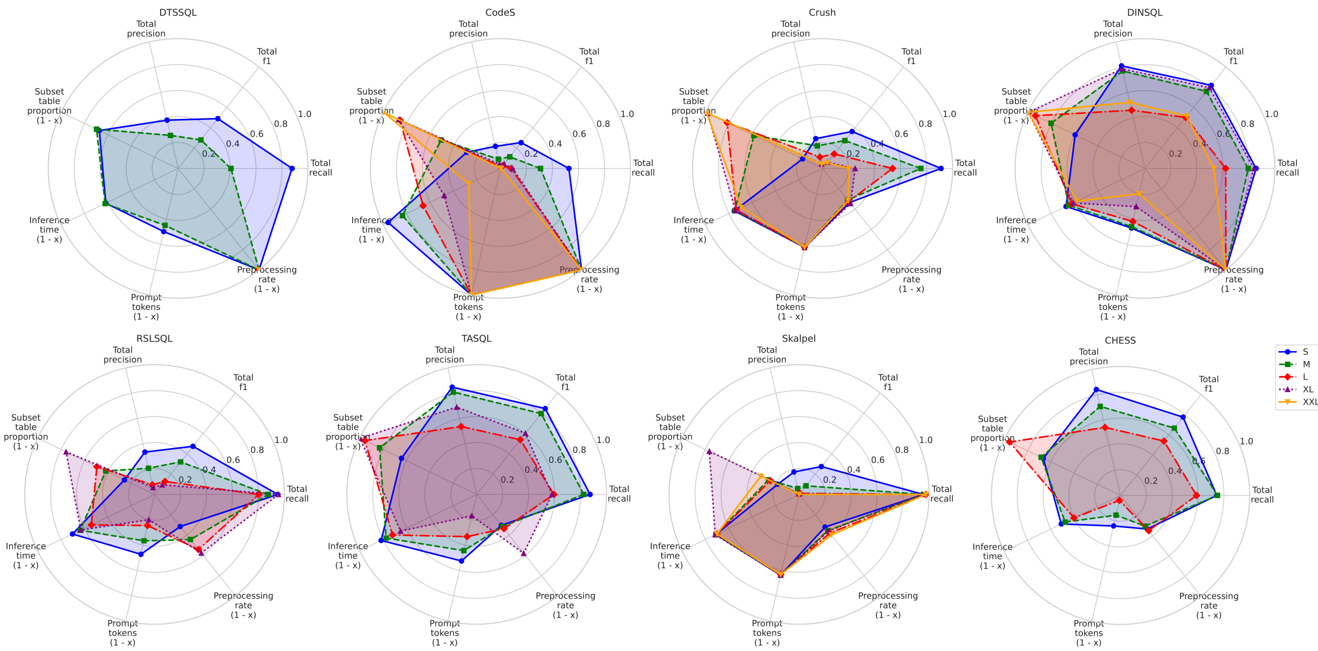

Figure 6: These radar charts display the tradeoffs between performance (measured by precision, f1, and recall) and time and resource usage (measured by inference time, prompt tokens, and pre-processing). Sensitivity to database size (S, M, L, XL, XXL) varies by both subsetting method and measure, and generally performance across all measures decreases as database size increases. Metrics where lower is better (schema proportion, inference time, token usage, preprocessing rate) are inverted (1 - x). Inference time, prompt tokens, and preprocessing rate are fit to the range [0, 1] via natural log functions.

Key Takeaways. Recall is the bottleneck metric for the downstream NL-to-SQL task. Recall failures at the subsetting stage guarantee subsequent NL-to-SQL failure. Our SKALPEL subsetter (by design) outperforms all other methods in this category for all schema sizes. However, recall is not the only value that affects downstream NL-to-SQL performance, and where SKALPEL shows weak results (precision and schema proportion), the LLM-based DINSQL and TASQL methods compete for precision and schema proportion dominance across all schema size categories. At the end of experiment 1, though we can clearly see which methods perform best in each metric category, we are left asking the question: *how will the differences across these metrics actually affect NL-to-SQL performance?* The answer to this question, which we address in experiment 2, will indicate which of these methods is *the best* performer in various contexts.

Another key observation is that method complexity does not guarantee improved results. In fact, we see the opposite. The highly complex CHESS and RSLSQL subsetters use orders of magnitude more tokens than, and are outperformed by, the simpler LLM-based methods DINSQL and TASQL.

Finally, SLM-based subsetters (CodeS and DTSSQL) never achieve parity with the LLM-based and hybrid methods in any metric except inference time and token efficiency. Their inability to handle large schemas (DTSSQL) or achieve usable results (CodeS) in terms of recall and precision remove them from consideration as viable candidates for improving NL-to-SQL, especially over large schemas.

Results Discussion. Frontier LLM-based methods including DINSQL, TASQL, CHESS, and RSLSQL generally outperform fine-tuned SLM- (CodeS, DTSSQL) and semantic search-based methods (Crush) by a significant margin in performance measures of f1, precision, and recall. However, the rapid growth of token usage across schema sizes exhibited by CHESS, RSLSQL, and TASQL portray the risk of over-reliance on knowledge enrichment and granular evaluation strategies. Knowledge enrichment, where schema representations are appended with sample values and verbose descriptions, cause both TASQL and RSLSQL to exceed the LLM's 1 million token context window for schemas in the XXL category. Granular column evaluation employed by CHESS, where each column is evaluated in isolation, incurs a very high token and time cost for relatively small schemas. The cost and time required to scale to XL and XXL schema sizes demands more than 10 million tokens per **Q**, and we consider it infeasible in terms of cost and time.

Recall is an especially critical metric for evaluating the potential usefulness of a subsetting method. Failure to recall required identifiers all but guarantees that a subsequent NL-to-SQL translation will fail to produce a correct answer. Only SKALPEL consistently produces high recall scores across all schema sizes , and this is done at the expense of reduced precision. Additionally, recall rates drop significantly for higher schema sizes in the XXL category with the best performing subsetting method (DINSQL) scoring a mean schema recall of 0.53. In contrast, our SKALPEL subsetter achieves a recall score of 0.98 because is tuned to maintain a high recall at the expense of lower precision.

Table 3: Experiment 1 The mean of subsetting performance metrics for all $\mathbb{S}'$ by schema size (Sz) and subsetting and subsetting function Ψ. Metrics include Inference Time (T (s)), Prompt Token Count (Tokens), Perfect Recall (PRe), Schema Recall (SchRe), Schema Precision (SchPr), Schema f1 (Schf1), Relation Recall (RelRe), Relation Precision (RelPr), Relation f1 (Relf1), Attribute Recall (AtrRe), Attribute Precision (AtrPr), Attribute f1 (Atrf1), Relation Proportion (RelPn), and Attribute Proportion (AtrPn). Section 4.2 provides definitions for each listed metric.

Sz	Ψ	T (s)	Tokens	PRe	SchRe	SchPr	Schf1	RelRe	RelPr	Relf1	AtrRe	AtrPr	Atrf1	RelPn	AtrPn
S	CHESS	31.85	229825	0.26	0.75	0.84	0.78	0.88	0.89	0.87	0.69	0.82	0.73	0.34	**0.10**
	CodeS	**0.27**	0	0.20	0.53	0.17	0.25	0.66	0.33	0.42	0.47	0.14	0.21	0.71	0.42
	Crush	3.06	425	0.64	0.91	0.24	0.36	0.98	0.39	0.53	0.88	0.20	0.32	0.83	0.49
	DINSQL	5.89	4988	0.50	0.86	0.81	0.82	0.94	0.86	0.89	0.82	0.79	0.79	0.40	0.13
	DTSSQL	9.42	3582	0.76	0.88	0.38	0.49	0.87	**0.92**	0.88	0.88	0.33	0.43	**0.33**	0.38
	RSLSQL	4.52	8836	0.86	0.95	0.33	0.47	**0.99**	0.49	0.62	0.93	0.31	0.44	0.74	0.35
	SKALPEL	3.61	**342**	**0.91**	**0.96**	0.18	0.28	0.96	0.41	0.54	**0.96**	0.15	0.24	0.82	0.83
	TASQL	1.51	2288	0.65	0.87	**0.85**	**0.85**	0.93	0.91	**0.91**	0.85	**0.83**	**0.82**	0.36	0.13
M	CHESS	96.69	891506	0.35	0.75	0.71	0.67	0.91	0.78	0.77	0.68	0.69	0.63	0.32	0.04
	CodeS	**1.15**	0	0.08	0.31	0.07	0.11	0.42	0.16	0.22	0.26	0.06	0.09	0.49	0.12
	Crush	2.60	431	0.41	0.76	0.18	0.28	0.91	0.39	0.51	0.69	0.15	0.23	0.41	0.13
	DINSQL	8.17	5989	0.50	0.80	0.77	0.76	0.90	0.86	0.86	0.74	0.73	0.71	0.20	**0.03**
	DTSSQL	8.82	7049	0.23	0.41	0.26	0.28	0.40	0.54	0.45	0.41	0.22	0.26	0.30	0.10
	RSLSQL	10.08	65715	0.76	0.88	0.21	0.32	**0.96**	0.37	0.49	0.83	0.18	0.28	0.58	0.12
	SKALPEL	3.73	**363**	**0.93**	**0.96**	0.05	0.09	**0.96**	0.24	0.36	**0.95**	0.04	0.07	0.74	0.80
	TASQL	2.76	9096	0.57	0.82	**0.81**	**0.80**	0.90	**0.94**	**0.91**	0.79	**0.76**	**0.75**	**0.17**	**0.03**
L	CHESS	351.95	5254439	0.31	0.59	**0.54**	**0.54**	0.74	0.63	0.66	0.51	**0.49**	**0.48**	0.05	**1e-3**
	CodeS	6.33	0	0.02	0.08	0.02	0.03	0.13	0.04	0.06	0.07	0.01	0.02	0.14	0.02
	Crush	**3.00**	441	0.24	0.54	0.09	0.14	0.82	0.19	0.30	0.41	0.07	0.11	0.19	0.02
	DINSQL	10.25	11566	0.43	0.62	0.46	0.50	0.71	0.54	0.59	0.56	0.42	0.45	0.06	**1e-3**
	RSLSQL	27.74	191906	0.70	0.80	0.08	0.13	0.93	0.35	0.40	0.73	0.05	0.09	0.51	0.05
	SKALPEL	4.16	**380**	**0.98**	**0.98**	0.01	0.01	**0.98**	0.07	0.12	**0.96**	1e-3	0.01	0.75	0.82
	TASQL	4.97	47440	0.32	0.60	**0.54**	**0.54**	0.74	**0.69**	**0.70**	0.51	0.47	0.47	**0.04**	**1e-3**
XL	CodeS	80.50	0	0.00	0.09	0.03	0.05	0.21	0.09	0.11	0.06	0.02	0.03	0.13	1e-3
	Crush	**2.58**	423	0.04	0.25	0.04	0.06	0.34	0.05	0.08	0.22	0.03	0.05	0.02	1e-3
	DINSQL	11.43	75973	0.46	0.84	**0.79**	**0.80**	0.94	**0.83**	**0.86**	0.80	**0.77**	**0.77**	**1e-3**	**1e-4**
	RSLSQL	15.91	391343	0.84	**0.95**	0.05	0.10	**0.98**	0.05	0.09	**0.94**	0.06	0.11	0.24	0.01
	SKALPEL	3.44	**338**	**0.88**	0.94	0.01	0.01	0.94	0.03	0.05	**0.94**	1e-3	0.01	0.23	0.23
	TASQL	6.87	644999	0.20	0.58	0.69	0.60	0.70	0.76	0.71	0.53	0.66	0.56	**1e-3**	**1e-4**
XXL	CodeS	381.43	0	0.00	0.01	1e-3	1e-3	0.01	1e-3	1e-3	1e-3	1e-3	1e-3	**1e-3**	1e-4
	Crush	4.49	431	0.03	0.20	0.03	0.06	0.36	0.04	0.07	0.15	0.03	0.05	0.01	1e-4
	DINSQL	20.79	384137	0.34	0.53	**0.52**	**0.52**	0.57	**0.57**	**0.56**	0.51	**0.50**	**0.50**	**1e-3**	**1e-5**
	SKALPEL	**4.02**	**370**	**0.96**	**0.98**	1e-4	1e-4	**0.97**	1e-3	0.01	**0.98**	1e-4	1e-4	0.67	0.67

Setting aside the nearly terminal effect of low recall, we also want methods to reduce the schema proportion as much as possible through high precision. TASQL's preliminary SQL parsing and identifier extraction approach yields the best schema precision in the S, M, L categories. For the XL, and XXL category where TASQL is infeasible due to context window limitations, DINSQL achieves the highest schema precision scores.

Schema Size Effects on Performance. As schema column and table counts increase, all subsetting methods demonstrate a reduced performance across all performance measures which means that larger schemas lead to more expensive, more time consuming, and less useful results. The Schf1 (Schema f1) column in Table 3 reveals a *schema f1* performance reduction for each subsetting method as schema sizes increase. Additionally, Figure 6 radar charts portray consistent performance reduction for larger schemas across both performance and resource use metrics.

Recall vs. Precision Trade-Off. Intuitively, there should be a trade-off between precision and recall, where subsetting methods that prioritize precision run an increased risk of false negatives and methods that prioritize recall would likely reduce precision to maximize the chance of true positives. For a hybrid method that uses semantic search (e.g., cosine similarity), then the selection of a distance threshold creates a natural Pareto frontier between recall and precision, as can be seen in our SKALPEL ablation analysis (see Figure 4). However, pareto frontiers across subsetting approaches (e.g., LLM, hybrid, and SLM) are less obvious, though there is some

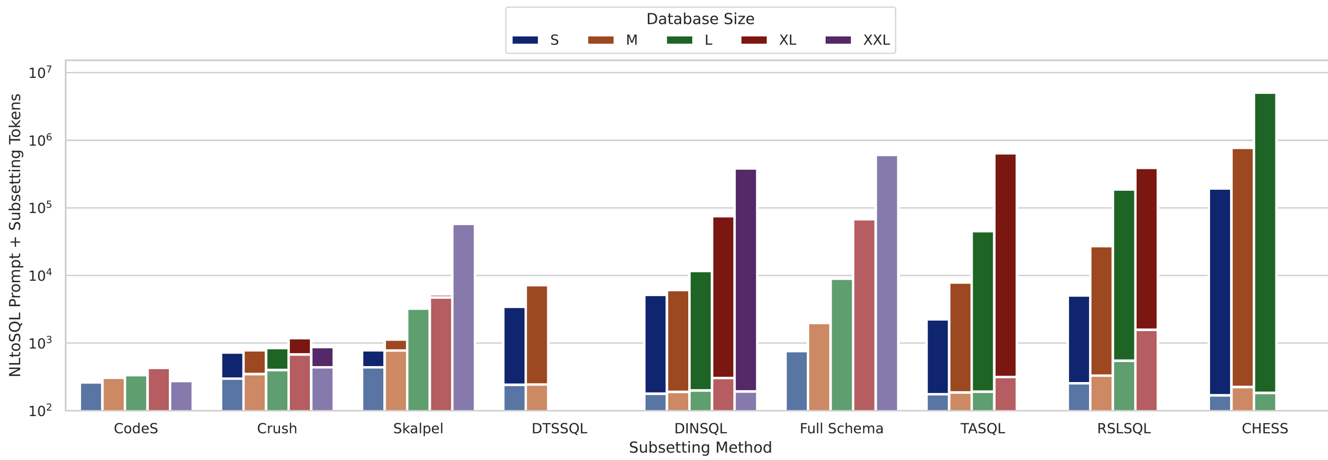

Figure 7: Total token usage along the Y-axis (log_{10} of prompt tokens) with the lower bar representing variations in the subset size and associated NL-to-SQL prompt tokens, and the upper bar representing tokens required for subset generation for each subsetting method and database size category. Except for Crush and SKALPEL, the subsetting process accounts for the significant majority of tokens in the combined subsetting-to-NL-to-SQL pipeline.

indication that LLM-based methods can tend to improve precision at a cost to recall.

Token Usage and Inference Time. Token usage is a cost driver for LLM-based systems either in terms of costs incurred through API transactions with LLM providers or as a surrogate for power consumption in self-hosted LLMs. Thus, the economnic use of LLMs necessitates the monitoring of token usage and motivates the discovery of methods that achieve performance objectives while minimizing token requirements.

In Figure 7 we see that token usage varies considerably by method, and all LLM-based methods consume more tokens than simply using the full schema for NL-to-SQL over small and medium databases. DINSQL is the sole LLM-based method that consumes fewer tokens for large, and larger, databases, whereas other LLM-based methods continue increasing in excess of full schema NL-to-SQL prompting. For example, CHESS requires between 263x (small schemas) and 537x (large schemas) more tokens than a full schema NL-to-SQL prompt. RSLSQL increases token usage from 4x (XXL schemas) to up to 27x (medium schemas).

The most token-efficient LLM-based methods are the Crush subsetting method and our own SKALPEL prototype which use an LLM only to either hallucinate a schema or decompose an NL question without referencing the target database schema. While both of these hybrid methods do not exhibit a subsetting prompt token growth correlation with database size, token usage from resulting higher schema proportions and subsequent larger NL-to-SQL prompts does cause some token usage increase during NL-to-SQL inference. Nevertheless, both Crush and SKALPEL consume fewer tokens than all LLM-based methods as well as full schema NL-to-SQL prompting.

5 EXPERIMENTS: NL-TO-SQL PERFORMANCE

Experiment 1 reveals that SKALPEL maximizes recall while sacrificing precision, whereas LLM-based methods do the opposite. Yet recall without downstream accuracy is meaningless. Experiment 2 tests whether these subsetting trade-offs translate to measurable improvements in SQL generation quality. To this end, we ask the following research questions:

Research Question 1. To what extent (if any) do schema subsetting methods improve NL-to-SQL execution accuracy?

Research Question 2. To what extent (if any) does the size of a schema affect the usefulness of schema subsetting?

Evaluation Method. The subsets $\mathbb{S}'_{Pred}$ generated in experiment 1 are the input to the NL-to-SQL generation in experiment 2. We evaluate the performance of $Q_{sqlPred}$ generated using prompts containing $\mathbb{S}'_{Pred}$ from all subsetting functions Ψ and measure them in terms of execution accuracy to determine the effect of schema subsetting outputs on the objective NL-to-SQL process.

Evaluation Metrics. We evaluate NL-to-SQL in terms of execution result set matching, typically referred to as *execution accuracy*. Execution accuracy is the comparison of the results r_{Gold} returned from a $Q_{sqlGold}$ query over $\mathbb{D}$ and r_{Pred} returned from a $Q_{sqlPred}$ query over $\mathbb{D}$. We adopt a subset-set comparison approach that accounts for the possibility that predicted queries may contain additional attributes not required in the gold query, but to not render the predicted query incorrect.

$$\forall \mathcal{A}_{gold} \in r_{Gold} \exists \mathcal{A}_{pred} \in r_{Pred} : (\mathbf{V} \in \mathcal{A}_{gold}) = (\mathbf{V} \in \mathcal{A}_{pred}) \tag{3}$$

That is, every attribute in $Q_{sqlGold}$ must be present in the $Q_{sqlPred}$, and the values in each attribute of the predicted query must equal their corresponding attribute values in $Q_{sqlGold}$.

5.1 NL-to-SQL Performance Evaluation Results

Key Takeaways. Subsetting method performance depends on LLM-type and schema size. Additionally, competing priorities of

Table 4: Mean of NL-to-SQL execution accuracy for each model category and schema size. Bold values indicate the highest result for each LLM category and schema size category. The '-' values indicate a category that the subsetting method is unable to process.

LLM Category	Economy					Flagship					Open				
Schema Size Category	S	M	L	XL	XXL	S	M	L	XL	XXL	S	M	L	XL	XXL
Perfect (Oracle)	0.59	0.50	0.42	0.41	0.62	0.64	0.53	0.48	0.45	0.66	0.61	0.52	0.46	0.37	0.64
SKALPEL	0.47	**0.41**	0.19	0.25	0.09	0.54	0.48	0.32	0.29	0.29	0.49	**0.45**	0.20	0.27	0.05
RSLSQL	0.50	0.40	**0.29**	0.30	-	0.56	0.47	0.39	0.36	-	0.51	0.41	**0.29**	0.32	-
TASQL	0.49	0.39	0.25	0.17	-	0.53	0.42	0.26	0.21	-	0.50	0.39	0.22	0.16	-
DINSQL	0.48	0.38	0.25	**0.33**	**0.27**	0.53	0.43	0.26	0.36	0.31	0.50	0.39	0.26	**0.33**	**0.28**
CHESS	0.33	0.30	0.23	-	-	0.44	0.37	0.22	-	-	0.37	0.32	0.21	-	-
Crush	0.40	0.27	0.11	0.04	0.02	0.47	0.33	0.16	0.07	0.05	0.42	0.28	0.12	0.04	0.03
CodeS	0.19	0.08	0.03	0.00	0.01	0.30	0.18	0.02	0.07	0.03	0.20	0.09	0.02	0.00	0.01
DTSSQL	0.46	0.08	-	-	-	0.52	0.13	-	-	-	0.47	0.06	-	-	-
FullSchema	**0.51**	**0.41**	0.19	0.26	0.04	**0.58**	**0.49**	**0.40**	**0.37**	**0.38**	**0.52**	0.44	0.16	0.29	0.0

Table 5: Logistic regression coefficients, standard error, and significance (p value) for non-colinear parameters with execution accuracy as the dependent variable, which is the result of all SQL queries generated by all LLMs from the subsets generated by all subset variations (n = 183,825), Pseudo R-squared = 0.1813. Execution accuracy is binary (1, 0) where 1 indicates a correct $Q_{sqlPred}$.

Metric	Coef	StdErr	pValue
Const	-3.539	0.024	0.000
TableRecall	0.251	0.044	0.000
TablePrecision	-0.087	0.027	0.001
ColumnRecall	3.382	0.037	0.000
ColumnPrecision	0.096	0.026	0.000
LogSubsetColumnCount	-0.429	0.008	0.000

maximizing execution accuracy and minimizing token expenditures will determine *a best* subsetting method for a particular use case.

For most LLM categories performing NL-to-SQL over *small* and *medium* schemas, subsetting degrades execution accuracy performance. This degradation, coupled with increased token usage compared to full-schema based NL-to-SQL for LLM-based subsetting methods, suggests that simply avoiding subsetting and instead using the full schema for *small* and *medium* sized schemas is the optimal approach for both accuracy and efficiency.

Full schema-based generation (no subsetting) is also optimal in terms of execution accuracy in all cases for all schema size categories when using the *flagship LLMs* GPT 4.1 and Gemini 2.5-pro.

Subsetting effectiveness when paired with economy and open-source models varies based on method and schema size. The LLM-based RSLSQL scores highest on execution accuracy over large-sized schemas with a significant token increase of 19x full schema token usage, while the LLM-based DINSQL outperforms all other methods over XL-, and XXL-sized schemas and also reduces token usage for XL (-19 percent difference), and XXL (-27 percent difference) compared to full schema usage.

For use cases where token efficiency is more important than maximizing execution accuracy, SKALPEL emerges as a strong contender, performing within a similar range as full schema NL-to-SQL across most LLM and schema size categories while achieving between -46 percent (medium schemas) to -90 percent (XXL schemas) percent decreases in token usage compared to the full schema.

Subsetting Metric Effects on Execution Accuracy. Table 5 presents logistic regression results correlating subsetting metrics from Experiment 1 (recall, precision, and subset size) against NL-to-SQL execution accuracy. The analysis uses non-colinear parameters only; we exclude F1 by covariance analysis and measure subset size in absolute terms (log column count) to isolate absolute size effects independent of upstream schema size.

The regression reveals a clear hierarchy of metric importance. *Table and column recall* dominate the effect on execution accuracy (coefficients of 0.251 and 3.382 respectively, both $p < 0.001$), confirming that omitting required identifiers during subsetting nearly guarantees downstream NL-to-SQL failure. In contrast, precision metrics show weaker effects: table precision carries a small negative coefficient (-0.087), and column precision a small positive one (0.096). Most surprisingly, the negative and substantial coefficient on LogSubsetColumnCount (-0.429) indicates that absolute subset size matters more than precision alone, and smaller subsets improve accuracy even when they contain false positives.

This metric hierarchy explains the performance tradeoffs observed in Experiment 1. SKALPEL maximizes recall at the cost of lower precision and larger subsets, yet Table 5 suggests that recall dominates accuracy outcomes. Conversely, LLM-based methods (DINSQL, TASQL) achieve higher precision but sacrifice recall; the regression shows this precision gain can be outweighed by recall losses. The negative *LogSubsetColumnCount* coefficient suggests that there is an absolute subset size penalty. Thus, the subsetter that maintains a high recall while minimizing subset size would theoretically maximize execution accuracy.

6 DISCUSSION AND LIMITATIONS

Subsetting Usefulness. Contemporary research expresses mixed results around the usefulness of schema subsetting, and in our work we expose some of the underlying reasons for these inconstencies: namely that subsetting performance–as well as downstream NL-to-SQL inference–is schema size and LLM type dependent, where schema size plays a large role in subsetting effectiveness and usefulness. From our analysis, we determine that subsetting small and medium schemas (less than 1,000 columns) actually tends to worsen NL-to-SQL performance for all types of LLMs.

Subsetting becomes useful when dealing with larger schemas and economy or open-source LLMs. This is because we see that for both open source and economy LLMs NL-to-SQL execution accuracy improves when using the LLM-based subsetters (e.g., RSLSQL, and DINSQL). In these cases, the decision to subset, and which subsetter to use, is more nuanced and depends on the tradeoff between performance and token usage. The simplest LLM-based approach (DINSQL) yields the best accuracy for XL and XXL schemas and reduces overall token usage. Thus, it may be beneficial to employ LLM-based subsetting when using low cost and open source LLMs for NL-to-SQL translation. Alternatively, our prototype SKALPEL hybrid subsetter offers a solution that has the potential to reduce token usage while maintaining execution accuracy at generally the same level as full schema NL-to-SQL.

Limitations. Although, to our knowledge, this is the first work to evaluate subsetting over very large database schemas, the majority of L-, XL-, and XXL-sized schemas come from the Spider 2 benchmark, which only comprises 221 of the 2,258 total NL-to-SQL questions across all 3 benchmarks. SNAILS provides 100 additional NL-to-SQL questions over its XXL-sized database. To remedy the shortage of NL-to-SQL questions over large schemas, we introduce the BigBird dataset with an additional 1,500 question-query pairs. While this greatly increases the number of questions over large schemas, it is likely that this approach is affected by the inevitable inclusion of references to the Bird [12] dev dataset in recent LLM training corpora. Future research into subsetting over very large schemas would benefit from a new and increased question pool of NL-SQL question query pairs that are absent from SoTA LLM training corpora.

Future Research. Even with additional measures to improve semantic matches between natural language questions and database schemas, we find that the SKALPEL semantic search-based method, while greatly improving token efficiency, at best approximates full-schema NL-to-SQL execution accuracy on large schemas.

LLM-based subsetting methods are still subject to context window constraints of the LLMs they use. We ask, would implementing a multi-pass context window-informed schema partitioning scheme with managed context and recall summarization be a viable approach to overcoming this token count problem?

To offset the disadvantages of expanding well-known NL-to-SQL benchmark datasets, a logical next step in this area of research is to expand the newer SNAILS [14] benchmark dataset to test LLM-based subsetting methods on data less likely to be included in their training corpora.

7 RELATED WORK

CRUSH [9] is a recent work that performs subsetting evaluation of a novel halucination-based subsetting method and compares it to variations of dense passage retrieval. The authors provide benchmark datasets including SpiderUnion, and BirdUnion–unions of the disjoint schemas in the Spider [31] and Bird [12] benchmarks respectively. They also introduce SocialDB, a collection of database schemas and schema descriptions without associated database instances and NL-SQL question-query pairs. In our work, we extend this line of research further by adopting the SNAILS collection to better-represent real-world schema subsetting challenges, and we adopt additional subsetting metrics. We also evaluate real-world subsetters from competitive NL-to-SQL systems.

A recent work evaluates the usefulness of schema subsetting using SoTA LLMs suggests that as LLM capability increases, the need for schema subsetting diminishes [16]. In this paper, the authors evaluate four LLM-based schema subsetting methods using the Bird SQL benchmark and measure performance using execution accuracy, false positive rates during subsetting, and schema linking recall. While related, our work provides additional measures of recall, precision, and F1 for various combinations of table and column identifiers. We also extend the scope of subsetting evaluation to non-LLM-based subsetting methods such as semantic similarity search- and finetuned classifier-based approaches. In addition to the Bird benchmark, we also evaluate SoTA linking methods using the Spider2 and SNAILS benchmarks.

Katsogiannis-Meimarakis, et al. [8] concurrently and independently developed a schema subsetting (or schema linking) evaluation methodology focused on LLM-based schema linking approaches in which they replicate several LLM-prompting methods for schema linking and evaluate them in terms of precision, recall, and accuracy using the Spider and Bird benchmarks [12, 31]. Our work complements this approach and makes use of the Spider 2 and SNAILS [10, 14] benchmarks which contain very large schemas. We also opt to reproduce existing subsetting methods using original code, and additionally evaluate subsetting performance in terms of token usage as well as both online inference and offline pre-processing time requirements.

ACKNOWLEDGMENTS

We thank the members of the UC San Diego Database Lab, Dr. Jingbo Shang, and Dr. Zhiting Hu for their feedback on this work. AI systems, including various GPT family models, were used for general editing and grammar review as well as for LaTeX formatting assistance. This work was supported in part by the U.S. Army Advanced Civil Schooling program and completed in part at the U.S. Army Cyber Institute by the first author.

REFERENCES

[1] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. arXiv:2411.00073 [cs.CL] https://arxiv.org/abs/2411.00073

[2] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306 [cs.CL]

[3] Yusuf Denizay Dönder, Derek Hommel, Andrea W Wen-Yi, David Mimno, and Unso Eun Seo Jo. 2025. Cheaper, Better, Faster, Stronger: Robust Text-to-SQL without Chain-of-Thought or Fine-Tuning. arXiv:2505.14174 [cs.CL] https://arxiv.org/abs/2505.14174

[4] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599 [cs.AI] https://arxiv.org/abs/2411.08599

[5] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xieand Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y.K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. https://arxiv.org/abs/2401.14196

[6] Kelly Hong, Anton Troynikov, and Jeff Huber. 2025. *Context Rot: How Increasing Input Tokens Impacts LLM Performance.* Technical Report. Chroma. https://research.trychroma.com/context-rot

[7] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models?. In *First Conference on Language Modeling*. https://openreview.net/forum?id=kIoBbc76Sy

[8] George Katsogiannis-Meimarakis, Katsiaryna Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. 2026. In-depth Analysis of LLM-based Schema Linking. In *Proceedings of the 29th International Conference on Extending Database Technology (EDBT 2026) (Advances in Database Technology)*, Vol. 29. Tampere, Finland, 117–130.

[9] Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. 2023. CRUSH4SQL: Collective Retrieval Using Schema Hallucination For Text2SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 14054–14066. https://doi.org/10.18653/v1/2023.emnlp-main.868

[10] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763* (2024).

[11] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3, Article 127 (May 2024), 28 pages. https://doi.org/10.1145/3654930

[12] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Chenhao Ma, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL]

[13] Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. 2026. XiYan-SQL: A Novel Multi-Generator Framework for Text-to-SQL. *IEEE Transactions on Knowledge and Data Engineering* (2026), 1–14. https://doi.org/10.1109/TKDE.2026.3657851

[14] Kyle Luoma and Arun Kumar. 2025. SNAILS: Schema Naming Assessments for Improved LLM-Based SQL Inference. *Proc. ACM Manag. Data* 3, 1, Article 77 (Feb. 2025), 26 pages. https://doi.org/10.1145/3709727

[15] Kyle Luoma and Arun Kumar. 2026. *Technical Report: A Comparative Evaluation of Schema Subsetting for LLM-based NL-to-SQL over Large-Schema Databases.* Technical Report. La Jolla, CA, USA. https://github.com/ADALabUCSD/ADALabUCSD.github.io/blob/master/papers/TR_2026_Schema_subsetting_eval.pdf

[16] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. https://doi.org/10.48550/ARXIV.2408.07702 arXiv:2408.07702

[17] OpenAI. 2025. gpt-oss-120b and gpt-oss-20b Model Card. arXiv:2508.10925 [cs.CL] https://arxiv.org/abs/2508.10925

[18] OpenAI. 2025. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/ Accessed: 2025-05-02.

[19] OpenAI. 2025. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/ Accessed: 2025-09-23.

[20] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[21] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. arXiv:2402.01117 [cs.CL]

[22] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. arXiv:2405.15307 [cs.CL] https://arxiv.org/abs/2405.15307

[23] Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. 2025. Automatic Metadata Extraction for Text-to-SQL. arXiv:2505.19988 [cs.DB] https://arxiv.org/abs/2505.19988

[24] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755

[25] Gemini Team. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL]

[26] Llama Team. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[27] PGVector Team. 2021. PGVector: Open-source vector similarity search for Postgres. https://github.com/pgvector/pgvector/. Accessed: 2025-09-25.

[28] Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, Rohan Anil, Ethan Dyer, Siamak Shakeri, Roopali Vij, Harsh Mehta, Vinay Ramasesh, Quoc Le, Ed Chi, Yifeng Lu, Orhan Firat, Angeliki Lazaridou, Jean-Baptiste Lespiau, Nithya Attaluri, and Kate Olszewska. 2024. Michelangelo: Long Context Evaluations Beyond Haystacks via Latent Structure Queries. arXiv:2409.12640 [cs.CL] https://arxiv.org/abs/2409.12640

[29] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL]

[30] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. arXiv:2502.14913 [cs.CL] https://arxiv.org/abs/2502.14913

[31] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium.

[32] Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2025. Jasper and Stella: distillation of SOTA embedding models. arXiv:2412.19048 [cs.IR] https://arxiv.org/abs/2412.19048